A Dynamic Compressed Accessibility Map for Secure XML Querying and Updating

MEGHADAD MIRABI, HAMIDAH IBRAHIM, LEILA FATHI, NUR IZURA UDZIR AND ALI MAMAT

Department of Computer Science
Faculty of Computer Science and Information Technology
Universiti Putra Malaysia
Selangor, 43400 Malaysia
E-mail: {meghdad.mirabi; leila.fathi67}@gmail.com; {hamidah.ibrahim; izura; ali}@upm.edu.my

By specifying a fine-grained access control on the XML data, an accessibility map is required to determine the accessibility of XML nodes for a specific subject (e.g. user or role) under a specific action (e.g. read or write). In the recent years, several research works have been done to reduce the overall storage cost of accessibility map with rapid determination of accessibility of XML nodes at runtime but there is no effort to implement the accessibility map in a compact format for dynamic environment where the accessibility of XML nodes can be updated frequently. In this paper, we propose a Dynamic Compressed Accessibility Map called DCAM to implement the accessibility map in a compact format which can be used in dynamic environment. Moreover, we suggest an efficient lookup method to determine the accessibility of XML nodes by labeling the authorization nodes in the DCAM with the dynamic XML labeling scheme. We also propose an efficient method to accelerate the process of checking the access authorizations for a set of XML nodes retrieved from the XML query processor when the access locality among the XML nodes in the XML tree is high. Besides, we define a set of constraints on the process of XML updating in order to maintain the DCAM in a compact format with minimum maintenance cost. The experimental results demonstrate that the DCAM is more efficient in both the space and time requirements for secure XML querying and updating.

Keywords: access control, encoding and labeling scheme, privacy, XML querying, XML updating

1. INTRODUCTION

In relational database systems, when a user submits a query, all the entities such as tables and views referred to in the query are checked against the access authorizations granted to the user. If one of the entities is inaccessible to the user, the whole query is rejected by the system. This kind of access control is called coarse-grained access control. In contrast to relational database systems that grant the access authorizations at the granularity of the whole tables or views, in XML database systems, the access authorizations are specified at a finer granularity, e.g. XML nodes and each query is answered using that portion of XML data that is accessible to the user.

In general, an accessibility map determines whether a specific subject can access a specific object under a specific action or not [1-3]. The accessibility map is often represented as an access control matrix where the subjects, objects, and actions form three...
dimensions of the matrix. Each element of this matrix can be either accessible or inaccessible with respect to whether a specific subject can access a specific object under a specific action or not. Constructing such accessibility map supports rapid determination of accessibility of objects at runtime but incurs more storage and maintenance costs [1, 3]. The storage space of accessibility map is proportional to the product of the number of subjects, the number of objects, and the number of actions supported by the system. Therefore, one of the challenging issues related to specifying a fine-grained access control for the XML data is how to implement the accessibility map in a compact format.

By specifying a fine-grained access control on the XML data, the process of access authorization checking is required at runtime. Thus, the access control mechanism which checks the accessibility of XML node must be efficient [2, 4]. This is not generally an issue in relational database systems with a coarse-grained access control. In relational database systems, the access authorizations are checked once before processing the query. If all the entities on which a query depends on are accessible, the query is processed. Otherwise, it is rejected. However, in XML database systems, each XML query is answered based on the part of XML data that is accessible for the subject who submits the query.

Recently, several research works have been done to reduce the overall storage cost of accessibility map with rapid determination of accessibility of XML nodes at runtime [1-3]. The Compressed Accessibility Map (CAM) proposed by [1] compresses the accessibility map by exploiting the access locality among the XML nodes. In general, only a small portion of the accessibility map accounts for the total size of the CAM. The Integrated CAM (ICAM) proposed by [3] is an improvement over the original CAM which combines multiple CAMs into an ICAM. The correlation among different actions (e.g. if the action “write” is allowed, the action “read” is automatically allowed as well) are used by the ICAM to compress multiple CAMs into one integrated structure. As a result, by pursuing a defined operational hierarchy among different actions, the ICAM is capable to integrate the multiple CAMs into an ICAM. However, this is not always desirable. For example, without being capable of reading an XML document, a subject may write to it in a mandatory access control model. The Compact Bit String Accessibility Map (CBSAM) proposed by [2] implements the accessibility map in a compact format by exploiting the access locality among the XML nodes in the marked XML tree. In contrast to the CAM which constructs a CAM for each subject, the accessibility of XML nodes are combined together in the CBSAM to support multiple subjects. Although the CAM, the ICAM, and the CBSAM are able to compress the accessibility map but they are not able to be used in dynamic environment where the accessibility of XML nodes can be changed by inserting new nodes into and deleting existing nodes from the XML tree.

The aim of this paper is to implement the accessibility map in a compact format in such a way that it can be used in dynamic environment with less maintenance cost. Hence, the main contributions of this paper are summarized as follows:

- We propose a Dynamic Compressed Accessibility Map called DCAM to implement the accessibility map in a compact format. Similar to the CAM proposed by [1], the DCAM exploits the access locality among the XML nodes to compress the accessibility map but in contrast to the CAM, it can be used in dynamic environment.
- We devise an efficient lookup method for the DCAM which supports rapid determination of the accessibility of XML nodes at runtime. The efficiency of the lookup method
is obtained by labeling the authorization nodes in the DCAM as well as the XML nodes in the marked XML tree with the BS-Containment labeling scheme. Moreover, an efficient method is suggested to accelerate the process of access authorization checking for a set of XML nodes retrieved from the XML query processor when the access locality among the XML nodes is high.

- We define a set of constraints for the process of XML updating in order to maintain the DCAM as compact as possible with minimum maintenance cost.
- We verify the effectiveness of the DCAM in both space and time requirements for secure XML querying and updating by conducting several experiments.

The rest of the paper is organized as follows: in Section 2, previous researches on the XML access control are investigated. In Section 3, the DCAM which implements the accessibility map in a compact format is described. The BS-Containment labeling scheme which is used to label the XML nodes in the XML tree as well as the authorization nodes in the DCAM is explained in Section 4. In Section 5, the efficient storage and lookup methods for the DCAM are explained. In Section 6, an efficient method which can be used by the DCAM to accelerate the process of checking the access authorizations is explained. A set of constraints which must be applied to the process of XML updating in order to ensure the consistency and maintain the DCAM in a compact format in dynamic XML environment is defined in Section 7. The experimental results are illustrated in Section 8. In Section 9, the process of integrating the multiple DCAMs into an integrated structure is discussed. Finally, the paper is concluded in Section 10.

2. RELATED WORK

Several researches have been done to specify and enforce a fine-grained access control on the XML data which include studies [4-30]. Some of these research works [6-9, 12, 16, 28] have defined models to specify a fine-grained XML access control, generally using the discretionary access control or role based access control. These researches focus on issues such as the level of granularity which access authorization should be specified (e.g. DTD, XML document, and XML Elements), the propagation policies (e.g. local and recursive), the conflict resolution policies (e.g. the grant overrides and the most specific overrides), and the default policies (e.g. the closed policy and the open policy).

Moreover, several mechanisms have been proposed to enforce a fine-grained access control on the XML data. Some of these research works [6, 7, 9, 10, 15, 16, 20, 25] are view-based which generate and maintain the different XML views for the different users based on their defined access authorizations. These views are created at compile time and contain the set of XML nodes which the users have permission to access. At runtime, the users’ queries are submitted against these views without worrying about the access control enforcement. The limitation of view based XML access control mechanism is that it incurs high maintenance and storage cost since many views should be created for the large number of users with different access authorizations. Besides, the problem of view based approach in the process of XML updating is synchronization. During XML updating, all views which contain the related data must be updated. In order to avoid the
view materialization at compile time, the proposed mechanisms in [10, 15, 25] generate a DTD view which only represents accessible portion of the XML document to the users. At runtime, the users’ queries are translated to the safe queries based on the DTD view. However, these mechanisms also have the drawbacks of view-based access control mechanism.

In traditional node filtering mechanisms [6, 7, 9, 13, 14], the access authorizations are determined by labeling the XML nodes with a permission (+), or a denial (-) and then pruning the XML tree based on the associated signs. Therefore, such a mechanism requires the repetitive XML document labeling and pruning for each user’s query which degrade the query performance at runtime. Recently, an efficient XML access control mechanism integrated with query processing using DP (Dynamic Predicate) is devised in [4]. The accessibility of XML nodes in the proposed mechanism is checked during the query execution time using the DP. The key idea for integrating access control with XML query processing is to discover a set of XML nodes which have the same accessibility.

In query rewriting access control mechanisms [10, 11, 16-23, 26, 27, 30], the access authorizations are employed to rewrite the probable unsafe queries into the safe ones which should be evaluated against the original XML dataset. A safe query is a query which its result does not violate any access authorizations. Due to the need of runtime process to rewrite the queries, such a mechanism degrades the query performance. The main idea of static analysis proposed by [19], as a pre-processing access control mechanism, is to make automata for the XML queries, the XML access authorizations, and the XML schemas and then compare them. The static analysis is not intended to entirely eliminate runtime checking, but rather intended to complement it. When the static analysis cannot provide determinate answer, runtime checking is needed. This method classifies an XML query at compile time into three categories: entirely authorized, entirely prohibited, or partially authorized. The entirely authorized or entirely prohibited queries can be executed without access control. However, the static analysis cannot obtain any benefits when a query is classified as a partially authorized one. The QFilter proposed by [11, 27], as an external pre-processing XML access control system, checks the XPath queries against the access authorizations and rewrites the queries according to the access authorizations before passing the revised queries to the XML query engine to process. The static analysis method needs a runtime checking to filter out the unauthorized data while the QFilter solves this problem by rewriting the XPath queries to filter out the unauthorized part of queries before passing them to the XML query engine. Thus, the QFilter has much better performance than the static analysis method. However, if there are many access authorizations for each role, the NFA (Non deterministic Finite Automata) based approach in the QFilter may have unacceptable overhead. Moreover, the QFilter rewrites some queries incorrectly according to the examples derived in [17]. On the contrary, a DFA (Deterministic Finite Automata) based enforcement mechanism is devised in [23, 30] which decreases the complexity of query rewriting process and always checks whether the user has the right to access the nodes that occur within the predicates.

The XML access control mechanism proposed by [17, 18, 26] is based on access control abstraction and query rewriting. The access control abstraction is an efficient mechanism to check only the necessary access authorizations for the user’s query instead
of checking all of the access authorizations. Moreover, the process of query rewriting is based on extending or eliminating the XML nodes of DTD using operators such as union, intersection, and except. These operators are supported by the proposed mechanism to rewrite the queries into safe and correct queries without violating the access authorizations.

Refer to [31] for more information about these research works.

3. DYNAMIC COMPRESSED ACCESSIBILITY MAP (DCAM)

In general, an accessibility map can be defined through a function \( F: S \times O \times A \rightarrow \{\text{Accessible}, \text{Inaccessible}\} \) where \( S \) is a set of subjects (e.g. users or roles), \( O \) is a set of objects (e.g. XML nodes), and \( A \) is a set of actions (e.g. read and write). The main idea of compressing the accessibility map is to keep a small number of XML nodes with their accessibilities instead of explicitly keeping the list of XML nodes with their accessibilities.

Here, we refer an XML tree marked based on the accessibility map as a *marked XML tree*. A marked XML tree for a single subject and action is illustrated in Fig. 1. In Fig. 1, the gray rectangles represent accessibility of XML nodes to the subject while the white rectangles represent non-accessibility of XML nodes to the subject.

![Accessibility of Single Subject](image1)

![XML Nodes and Accessibility](image2)

Fig. 1. A marked XML tree with the corresponding CAM and DCAM.

Similar to the CAM proposed in [1], the DCAM exploits the access locality among the XML nodes in the marked XML tree in order to compact the accessibility map. The
access locality among the XML nodes means that the XML nodes clustered together have similar accessibility. The access locality can be horizontally only between sibling nodes or vertically between the parent nodes and children nodes in a marked XML tree.

**Definition of Authorization Node:** An authorization node is a node in the marked XML tree in which its accessibility is different from its parent node. Note that we assume that the root node of the marked XML tree is an authorization node.

The DCAM corresponding to a given marked XML tree is the set of authorization nodes together with their accessibilities when the marked XML tree is traversed in the preorder. For example, the DCAM corresponding to the marked XML tree in Fig. 1 only contains four authorization nodes \( A, L, D, \) and \( G \) with their accessibilities.

**Property 3.1:** Let \( AN(\text{DCAM}_{s_i, a_j}) \) be the set of authorization nodes in the DCAM of the subject \( s_i \) and the action \( a_j \) and \( |AN(\text{DCAM}_{s_i, a_j})| \) be the total number of authorization nodes in the DCAM of the subject \( s_i \) and the action \( a_j \). The total number of bits required to represent the accessibility of the authorization nodes in the DCAM is \( |AN(\text{DCAM}_{s_i, a_j})| \) bits.

**Property 3.2:** The time to construct the DCAM for a specific subject and action is proportional to the size of the marked XML tree (number of XML nodes in the marked XML tree).

In order to compare the DCAM with the CAM, here we explain the process of compressing the accessibility map in the CAM. The compression of the accessibility map achieved by the CAM relies on two observations. First, in a region of the XML tree, uniform accessibility of descendants of an XML node can be represented at the node itself. For instance, if each node in a sub-tree of the marked tree is accessible, substantial compression can be achieved if the CAM maintains the root node of the sub-tree with a \((d^+, s^+)\) label, indicating that the node is accessible \((s^+)\), and so are its descendants \((d^+)\). This observation can be generalized to employ a simple node labeling mechanism, involving \(d^+, d^-, s^+, \) and \(s^-\). The semantics of the labels are as follows. If node \( x \) carries an \( s^+ \) (respectively \( s^- \)), then \( x \) is accessible (respectively, inaccessible). If node \( x \) carries a \( d^+ \) (respectively \( d^- \)), and \( y \) is a descendant of \( x \), then \( y \) is accessible (respectively, inaccessible), unless this is overridden by the label of a closer ancestor of \( y \) (or by the label at \( y \) itself). Second, it is frequently the case that, at least within a local region of the XML tree, there is a hierarchical aspect to accessibility: if a node is accessible, then so are its ancestors. We call this the **ancestor accessibility property**. This observation can be utilized in the CAM as follows: if a node \( x \) carries an \( s^+ \), then so must all its ancestors, within the local region. When the ancestor accessibility property does not hold over the entire marked XML tree, it is easy to partition the tree into regions that satisfy the ancestor accessibility property. We call each resulting XML sub-tree a **unit region**. The algorithm to partition a given marked XML tree into unit regions is very simple: Perform a traversal of the XML tree, and mark each node \( u \) that is accessible but the parent node of \( u \) is inaccessible. Each such node is called a **marker node**, since it demarcates the boundary of a unit region. The sub-tree rooted at a marker node (or at the root node of the entire XML tree) is a unit region, excluding any descendant marker nodes and their
associated sub-trees. For example, node $G$ is a marker node in the marked XML tree illustrated in Fig. 1.

Consider the CAM corresponding to the marked XML tree in Fig. 1 which only contains the XML nodes $A$, $L$, $D$, and $G$ with their labels. Node $B$ and its descendants except node $L$ can be inferred to be accessible on account of the $d^+$ label at their nearest labeled ancestor (i.e. node $A$). Node $A$ itself can be inferred to be accessible because of its own $s^+$ label. Node $L$ would have been considered inaccessible because of its own $s^-$ label. Node $C$ can be inferred to be accessible on account of the $d^+$ label at its nearest labeled ancestor (i.e. node $A$). Node $H$ and node $i$ and all their descendants are inaccessible on account of the $d^-$ label at their nearest labeled ancestor (i.e. node $D$). Node $D$ itself can be inferred to be inaccessible because of its own $s^-$ label. Node $G$ can be inferred to be accessible because of its own $s^+$ label and its descendants can be inferred to be accessible on account of the $d^+$ label at node $G$. Refer to [1] for more information about the CAM.

Property 3.3: Let $\mathcal{N}(\text{CAM}_{s_i,a_j})$ be the set of XML nodes in the CAM of the subject $s_i$ and the action $a_j$ and $|\mathcal{N}(\text{CAM}_{s_i,a_j})|$ be the total number of XML nodes in the CAM of the subject $s_i$ and the action $a_j$. The total number of bits required to represent the accessibility of the XML nodes in the CAM is $3 \times |\mathcal{N}(\text{CAM}_{s_i,a_j})|$ bits.

Note that we need three bits to represent the accessibility of each XML node in the CAM since the accessibility of each XML node in the CAM is represented as $(s^*, d^*)$ where $*$ can be $+$ or $-$. $s^+/s^-$ means that the current node is accessible/inaccessible and $d^+/d^-$ means that the descendant nodes of the current node are also accessible/inaccessible unless they are overruled by the label of their nearest ancestor nodes. Therefore, two bits are required to represent the accessibility of each XML node in the CAM. Moreover, one bit is needed to represent whether the XML node in the CAM is a marked node or not. The marker node is the root of a unit region in the CAM. In the CAM, the marked XML tree is partitioned into unit regions when a node is accessible but its ancestor nodes are not.

4. BS-CONTAINMENT LABELING SCHEME

In this section, we explain the BS-Containment labeling scheme [32] since it is used to label the authorization nodes in the DCAM and the XML nodes in the marked XML tree. In Section 4.1, we explain how the XML nodes are labeled using the BS-Containment labeling scheme. In Section 4.2, we explain how the BS-Containment labeling scheme can facilitate the process of XML querying. In Section 4.3, we explain how the BS-Containment labeling scheme can efficiently update the XML tree without affecting the order of the XML nodes.

4.1 The Process of XML Encoding and Labeling

Generally, the BS-Containment labeling scheme is a containment labeling scheme [33-39] which the start and end values of each XML node are encoded with the bit string codes. In the containment labeling scheme, each XML node is assigned with two values,
start and end, based on the positions of start and end tags of the node in the XML document. In this way, the ancestor-descendant relationship between two arbitrary nodes can be determined. Moreover, the level value is added to each node label in order to determine the parent-child relationship between two arbitrary XML nodes. For example, the XML tree in Fig. 2 is labeled using the containment labeling scheme. The problem of the containment labeling scheme occurs during the XML updating process when new nodes are inserted into the XML tree. In this case, many nodes have to be re-labeled in order to maintain the ordering of nodes in the XML tree. For example, if node α is to be inserted into the XML tree in Fig. 2, all the gray nodes need to be re-labeled. In order to solve this problem, the BS-Containment labeling scheme uses a fractional number based encoding scheme to encode the start and end values of each XML node. Then, these assigned fractional numbers are converted to the corresponding bit string codes to save the storage space as well as accelerate the process of XML querying and updating.

Fig. 2. Containment labeling scheme.

If the total number of nodes in the XML tree is |N|, then, the total number of fractional numbers required to encode the start and end values of the XML nodes in the BS-Containment labeling scheme is $2 \times |N|$. The following example illustrates how the fractional numbers can be assigned to a set of ordinal decimal numbers. The second column of Table 1 shows the fractional numbers assigned to 34 ordinal decimal numbers. We choose 34 as an example but generally the fractional numbers can be assigned to any set of ordinal decimal numbers.

**Example 4.1.1:** In order to assign the fractional numbers to a set of 34 ordinal decimal numbers, we assume that there is a number before 1 which is 0 and a number after 34 which is 35. Next, we assign the middle fractional number between (0, 1) to the middle decimal number between 0 and 35. The middle fractional number between (0, 1) is $\frac{1}{2}$ which is calculated by $(0 + 1) / 2$ and the middle decimal number between 0 and 35 is 18 which is calculated by $(0 + [(35 - 0) / 2])$. Therefore, $\frac{1}{2}$ is assigned to 18. Next, for the set of ordinal decimal numbers from 0 to 18, we assign the middle fractional number between (0, $\frac{1}{2}$) and for the set of ordinal decimal numbers from 18 to 35, we assign the
Table 1. Fractional numbers and bit string codes for 34 ordinal decimal numbers.

<table>
<thead>
<tr>
<th>Decimal Number</th>
<th>Fractional Number</th>
<th>Bit String Code</th>
<th>Decimal Number</th>
<th>Fractional Number</th>
<th>Bit String Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1/64</td>
<td>000001</td>
<td>18</td>
<td>1/2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1/32</td>
<td>00001</td>
<td>19</td>
<td>33/64</td>
<td>10001</td>
</tr>
<tr>
<td>3</td>
<td>1/16</td>
<td>0001</td>
<td>20</td>
<td>17/32</td>
<td>10001</td>
</tr>
<tr>
<td>4</td>
<td>3/32</td>
<td>00011</td>
<td>21</td>
<td>9/16</td>
<td>1001</td>
</tr>
<tr>
<td>5</td>
<td>1/8</td>
<td>001</td>
<td>22</td>
<td>19/32</td>
<td>10011</td>
</tr>
<tr>
<td>6</td>
<td>5/32</td>
<td>00101</td>
<td>23</td>
<td>5/8</td>
<td>101</td>
</tr>
<tr>
<td>7</td>
<td>3/16</td>
<td>0011</td>
<td>24</td>
<td>21/32</td>
<td>10101</td>
</tr>
<tr>
<td>8</td>
<td>7/32</td>
<td>00111</td>
<td>25</td>
<td>11/16</td>
<td>1011</td>
</tr>
<tr>
<td>9</td>
<td>1/4</td>
<td>01</td>
<td>26</td>
<td>23/32</td>
<td>10111</td>
</tr>
<tr>
<td>10</td>
<td>17/64</td>
<td>010001</td>
<td>27</td>
<td>3/4</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>9/32</td>
<td>01001</td>
<td>28</td>
<td>25/32</td>
<td>11001</td>
</tr>
<tr>
<td>12</td>
<td>5/16</td>
<td>0101</td>
<td>29</td>
<td>13/16</td>
<td>1101</td>
</tr>
<tr>
<td>13</td>
<td>11/32</td>
<td>01111</td>
<td>30</td>
<td>27/32</td>
<td>11011</td>
</tr>
<tr>
<td>14</td>
<td>7/8</td>
<td>11</td>
<td>31</td>
<td>7/8</td>
<td>111</td>
</tr>
<tr>
<td>15</td>
<td>15/32</td>
<td>01101</td>
<td>32</td>
<td>29/32</td>
<td>11101</td>
</tr>
<tr>
<td>16</td>
<td>7/16</td>
<td>00111</td>
<td>33</td>
<td>15/16</td>
<td>1111</td>
</tr>
<tr>
<td>17</td>
<td>15/32</td>
<td>00111</td>
<td>34</td>
<td>31/32</td>
<td>11111</td>
</tr>
</tbody>
</table>

middle fractional number between (½, 1). The middle fractional number between (0, ½) is ¼ (½ + 1) / 2 and the middle decimal number between 0 and 18 is 9 (0 + [(18 – 0) / 2]). Therefore, ¼ is assigned to 9. Moreover, the middle fractional number between (½, 1) is ¾ (½ + 1) / 2 and the middle decimal number between 18 and 35 is 27 (18 + [(35 – 18) / 2]). Therefore, ¾ is assigned to 27. We continue this process until all the ordinal decimal numbers from 1 to 34 are being processed.

In the BS-Containment labeling scheme, each generated fractional number in the form of $i/2^j$ is converted to a corresponding bit string code using the following mapping method in order to save the storage space and accelerate the process of XML querying and updating.

Mapping from Fractional Number to Bit String Code

Let $FN = \{i/2^j \mid 0 < i < 2^j, i = 2k – 1, k \in \mathbb{N}, j \in \mathbb{N} \}$ denotes a set of fractional numbers and $BS = \{\alpha \beta \mid \alpha \in \{0, 1\}, \beta = 1 \}$ denotes a set of bit string codes, a mapping function $M: FN \rightarrow BS$ is defined as follows:

$$M(i/2^j) = \begin{cases} \text{binary}(i) & \text{if length(binary}(i)) = i \\ \text{Pow}(j – \text{length(binary}(i)))\text{binary}(i) & \text{if length(binary}(i)) < j \end{cases}$$

where

- Function $\text{binary}(i)$ returns the binary code of $i$.
- Function $\text{length}(i)$ returns the length of binary code $i$.
- Function $\text{pow}(i)$ returns a bit string which comprises of $i$ occurrences of 0.
The third column of Table 1 shows the bit string codes that are assigned to the fractional numbers. For instance, the fractional number assigned to the decimal number 4 is 3/32 and thus $i/2^j$ is 3/32 ($i = 3, j = 5$). Therefore, the bit string code of 3/32 is $\text{pow}(5 - \text{length}(\text{binary}(3))) \text{ binary}(3) = 00011$ since $\text{binary}(3) = 11, \text{length}(\text{binary}(3)) = 2$, and $2 < j (= 5)$.

By replacing the start and end values of XML nodes in the containment labeling scheme with the corresponding bit string codes, the containment labeling scheme can be converted to the BS-Containment labeling scheme. For example, by replacing the start and end values of each XML node in Fig. 2 with the corresponding bit string codes in Table 1, an XML tree labeled by the BS-Containment labeling scheme is formed similar to Fig. 3.

As shown in Fig. 2, the level value is added to each node label in order to determine the parent-child relationship between two XML nodes in the containment labeling scheme. However, such information is sensitive in the dynamic XML environment because the level values of many nodes must be modified when a node is inserted into the XML tree as a parent or an ancestor node. Thus, in the BS-Containment labeling scheme, the parent’s start value is added to each node label instead of the level value. The parent’s start value needs more storage space than the level value but only the parent’s start value of a node is needed to be modified when a parent or an ancestor node is inserted into the XML tree. Similar to Fig. 1, in Fig. 3, the gray rectangles represent accessibility of XML nodes to the subject while the white rectangles represent non-accessibility of XML nodes to the subject. Moreover, in Fig. 3, the authorization nodes are shown with ★.

![Fig. 3. A marked XML Tree labeled by the BS-containment labeling scheme.](image)

In the BS-Containment labeling scheme, the bit string codes are compared based on the lexical order rather than the numerical order. The definition of lexical order is as follows:

**Definition of Lexicographical Order (<):** If $\text{LeftBitString}$ represents the left bit string and $\text{RightBitString}$ represents the right bit string, then
(1) *LeftBitString* is lexicographically equal to *RightBitString* if and only if they are the same bit by bit.

(2) *LeftBitString* is smaller than *RightBitString* (*LeftBitString* < *RightBitString*) if and only if at least one of the following conditions is satisfied:

(a) The current bit of *LeftBitString* is “0” and the current bit of *RightBitString* is “1”, if we compare them bit by bit from left to right.

(b) *LeftBitString* is the prefix of *RightBitString*.

For example, based on Table 1, the bit string codes assigned to three decimal numbers 6, 7, and 8 are “00101”, “0011” and “00111” respectively and “00101” < “0011” < “00111” lexicographically based on the definition of lexicographical order.

### 4.2 The Process of XML Querying

In order to facilitate the XML query processing, the structural relationships between two arbitrary XML nodes must be determined without directly access to the XML document. The BS-Containment labeling scheme is able to determine the structural relationships between two arbitrary XML nodes as well as support all XPath axes.

Given two XML nodes *a* and *B* labeled with (*starta*, *enda*, *parentStarta*) and (*startb*, *endb*, *parentStartb*) respectively, the structural relationships between nodes *a* and *b* based on the BS-Containment labeling scheme are determined as follows:

- **Ancestor-Descendant (A-D) Relationship**: node *a* is an ancestor of node *b* iff *starta* < *startb* and *enda* > *endb*. For example, in Fig. 3, node *D* is an ancestor of node *P* since “0111” < “11001” and “1111” > “1101”.

- **Parent-Child (P-C) Relationship**: node *a* is the parent of node *b* iff *starta* = *parentStartb*. For example, in Fig. 3, node *G* is the parent of node *M* since “01111” = “01111”.

- **Following Relationship**: node *a* is a following of node *b* iff *starta* > *endb*. For example, in Fig. 3, node *H* is the following of node *E* since “101” > “010001”.

- **Preceding Relationship**: node *a* is a preceding of node *b* iff *enda* < *startb*. For example, in Fig. 3, node *F* is the preceding of node *I* since “0101” < “1011”.

- **Following-Sibling Relationship**: node *a* is a following-sibling of node *b* iff *starta* > *startb* and *parentStarta* = *parentStartb*. For example, in Fig. 3, node *L* is the following-sibling of node *J* since “00111” > “00011” and “00011” = “00011”.

- **Preceding-Sibling Relationship**: node *a* is a preceding-sibling of node *b* iff *starta* < *startb* and *parentStarta* = *parentStartb*. For example, in Fig. 3, node *O* is the preceding-sibling of node *Q* since “10111” < “11011” and “1011” = “1011”.

### 4.3 The Process of XML Updating

In this section, we explain the behavior of the BS-Containment labeling scheme when XML nodes are deleted from or inserted into the XML tree. It should be mentioned that the BS-Containment labeling scheme is able to keep the order of the XML node in the XML tree even in the process of node insertion and deletion. It is proved in [32].

When a leaf node or a sub-tree is deleted from the XML tree, the process of re-labeling the pre-existing nodes in the XML tree is not required. In the case of internal node
deletion, only parent’s start value of its children must be changed to the start value of its parent while the start and end values of all nodes do not need to be modified. In the following sub-sections, we present the process of node insertion at different positions of the XML tree.

4.3.1 Generating an Inserted Bit String Code

The MiddleBitStringGenerator algorithm shown in Fig. 4 generates a new bit string code between two consecutive bit string codes. Note that the correctness of generating a new bit string code between two consecutive bit string codes using the MiddleBitStringGenerator algorithm is proved in [32].

MiddleBitStringGenerator (leftBitString, rightBitString, middleBitString)
Input: leftBitString and rightBitString such that leftBitString < rightBitString
Output: middleBitString such that leftBitString < middleBitString < rightBitString

1. BitStringToFractionalNumber(leftBitString, leftNominator, leftDenominator);
2. BitStringToFractionalNumber(rightBitString, rightNominator, rightDenominator);
3. middleNominator = (leftNominator × rightDenominator) + (leftDenominator × rightNominator);
4. middleDenominator = 2 × (leftDenominator × rightDenominator);
// gcd function returns the Greatest Common Denominator
5. gcdValue = gcd (middleNominator, middleDenominator);
6. middleNominator /= gcdValue;
7. middle Denominator /= gcdValue;
8. middle BitString = FractionalNumbertoBitString(middleNominator, middle Denominator);

/* BitStringToFractionalNumber procedure converts a bit string code to the corresponding fractional number and generates the nominator and denominator of fractional number */

BitStringToFractionalNumber (bitString, nominator, denominator)
Input: bit string code bitString
Output: nominator and denominator of fractional number

/* length function returns the length of a bit string code and decimal function returns the decimal number of a bit string code */
1. denominator = length(bitString);
2. nominator = decimal (bitString);

/* FractionalNumberToBitString function returns the bit string code of a fractional number */

FractionalNumberToBitString (nominator, denominator)
Input: nominator and denominator of fractional number

/* log2 function returns the logarithm in base 2 of a decimal number, length function returns the length of a bit string code, and binary function returns the binary code of a decimal number */
1. lengthDenominator = log2 (denominator);
2. for (int i = 0; i < (lengthDenominator - length (binary (nominator)))); i++)
3. bitString += '0';
4. bitString += binary (nominator);

Fig. 4. MiddleBitStringGenerator algorithm.
For example, for the \textit{leftBitString} “00111”, the \textit{leftNominator} and \textit{leftDenominator} are “7” and “32”, respectively and for the \textit{rightBitString} “01”, the \textit{rightNominator} and \textit{rightDenominator} are “1” and “4”, respectively based on the \textit{BitStringToFractional-Number} procedure. Now, based on the lines 3, 4, 5, 6, and 7 of the MiddleBitStringGenerator algorithm, the \textit{middleNominator} and \textit{middleDenominator} are “15” and “64”, respectively. Next, “001111” is generated by the \textit{FractionalNumberToBitString} function as the \textit{middleBitString}.

4.3.2 The process of node insertion

There are three kinds of insertions in the XML tree according to the positions in which nodes should be inserted: insertion of a node as a child of a leaf node, insertion of a node as a sibling node, and insertion of a node as a parent node.

The algorithm illustrated in Fig. 5 is proposed in [32] to insert a node as a child of a leaf node, \textit{targetNode}, labeled by (\textit{starttargetNode}, \textit{endtargetNode}, \textit{parentStarttargetNode}).

The BS-Containment labeling scheme assigns to each node label its parent’s start value instead of level value. Therefore, if the \textit{newNode} is labeled by (\textit{startnewNode}, \textit{endnewNode}, \textit{parentStartnewNode}), the following rules should be satisfied:

- \textit{starttargetNode} < \textit{startnewNode} < \textit{endnewNode} < \textit{endtargetNode}.
- \textit{parentStartnewNode} = \textit{StarttargetNode}.

For example, in Fig. 3, node $\alpha$ is to be inserted as a child of the leaf node $L$. Therefore, we have:

- $\textit{start}_\alpha = \text{MiddleBitStringGenerator}(00111, 01) = 001111$
- $\textit{end}_\alpha = \text{MiddleBitStringGenerator}(001111, 01) = 0011111$
- $\textit{parentStart}_\alpha = 00111$

![InsertChild (targetNode)](image)

**Fig. 5. InsertChild algorithm.**

The algorithm illustrated in Fig. 6 is proposed in [32] to insert a new node as the next sibling of \textit{targetNode}. For example, in Fig. 3, node $\beta$ is to be inserted after node $M$ as a new sibling node. Therefore, we have:

- $\textit{start}_\beta = \text{MiddleBitStringGenerator}(100001, 10001) = 1000011$
- $\textit{end}_\beta = \text{MiddleBitStringGenerator}(1000011, 10001) = 10000111$
- $\textit{parentStart}_\beta = 01111$
The process of inserting a new sibling node before a node is similar to the process of inserting a new sibling node after. Therefore, the explanation on this process is omitted here.

\[ \text{InsertSiblingAfter} (\text{targetNode}) \]

**Input:** targetNode labeled by \((\text{start}_{\text{targetNode}}, \text{end}_{\text{targetNode}}, \text{parentStart}_{\text{targetNode}})\)

**Output:** newNode labeled by \((\text{start}_{\text{newNode}}, \text{end}_{\text{newNode}}, \text{parentStart}_{\text{newNode}})\)

1. \(\text{next} = \) the start value of the nearest following-sibling of targetNode;
2. if there is not any following-sibling of targetNode then
3. \(\text{next} = \) the end value of targetNode’s parent;
4. \(\text{start}_{\text{newNode}} = \text{MiddleBitStringGenerator} (\text{end}_{\text{targetNode}}, \text{next});\)
5. \(\text{end}_{\text{newNode}} = \text{MiddleBitStringGenerator} (\text{end}_{\text{start}_{\text{newNode}}}, \text{next});\)
6. \(\text{parentStart}_{\text{newNode}} = \text{parentStart}_{\text{targetNode}};\)

Fig. 6. InsertSiblingAfter algorithm.

The algorithm illustrated in Fig. 7 is proposed in [32] to insert a new node as a parent of targetNode. The position of the new parent node of targetNode (newNode) is between the previous and the next sibling nodes of targetNode. Therefore, the start and end values of newNode are between the end value of the previous sibling node of targetNode and the start value of the next sibling node of targetNode. In case that the preceding-sibling (the following-sibling) of the targetNode does not exist, the start (end) value of targetNode’s parent can be used.

\[ \text{InsertParent} (\text{targetNode}) \]

**Input:** targetNode labeled by \((\text{start}_{\text{targetNode}}, \text{end}_{\text{targetNode}}, \text{parentStart}_{\text{targetNode}})\)

**Output:** newNode labeled by \((\text{start}_{\text{newNode}}, \text{end}_{\text{newNode}}, \text{parentStart}_{\text{newNode}})\)

1. \(\text{prev} = \) the end value of the nearest preceding-sibling of targetNode;
2. if there is not any preceding-sibling of targetNode then
3. \(\text{prev} = \) the start value of targetNode’s parent;
4. \(\text{next} = \) the start value of the nearest following-sibling of targetNode;
5. if there is not any following-sibling of targetNode then
6. \(\text{next} = \) the end value of targetNode’s parent;
7. \(\text{start}_{\text{newNode}} = \text{MiddleBitStringGenerator} (\text{prev}, \text{start}_{\text{targetNode}});\)
8. \(\text{end}_{\text{newNode}} = \text{MiddleBitStringGenerator} (\text{end}_{\text{targetNode}}, \text{next});\)
9. \(\text{parentStart}_{\text{newNode}} = \text{parentStart}_{\text{targetNode}};\)
10. \(\text{parentStart}_{\text{targetNode}} = \text{parentStart}_{\text{newNode}};\)

Fig. 7. InsertParent algorithm.

For example, in Fig. 3, node \(\gamma\) is to be inserted as the parent of node \(I\). Therefore, we have:

- \(\text{start}_{\gamma} = \text{MiddleBitStringGenerator} (10101, 1011) = 101011\)
- \(\text{end}_{\gamma} = \text{MiddleBitStringGenerator} (11101, 1111) = 111011\)
- \(\text{parentStart}_{\gamma} = 0111\)
- \(\text{parentStart}_{I} = 101011\)
In the containment labeling scheme, the level value is added into node label in order to determine the parent-child relationship between two XML nodes. Therefore, after inserting a node as a parent (an ancestor) node, the level value of all its descendants should be modified. But, in the BS-Containment labeling scheme, the parent’s start value is kept instead of level value. Hence, if a node is inserted as a parent (an ancestor) node, the parent’s start value of all its descendants is still unchanged except that of the child of the inserted node. Consequently, the BS-Containment labeling scheme needs only a node to be re-labeled when a new node is inserted as a parent (an ancestor) node.

Besides, the BS-Containment labeling scheme can handle sub-tree insertion without the need of re-labeling process. When a sub-tree is inserted into the XML tree, the BS-Containment labeling scheme can process sub-tree insertion as the insertion of nodes one by one.

5. EFFICIENT STORAGE AND LOOKUP METHODS FOR THE DCAM

We use the BS-Containment labeling scheme (i) to label the XML nodes in the marked XML tree as well as the authorization nodes in the DCAM since the BS-Containment labeling scheme is able to keep the order of XML nodes even when the XML tree is updated, (ii) to capture the required information to re-construct an ordered XML documents, and (iii) to determine the structural relationship between two arbitrary XML nodes.

The metadata of XML nodes in the marked XML tree labeled using the BS-Containment labeling scheme is stored into the BS-Containment-METADATA relation. The schema of this relation is as follows:

**BS-Containment-METADATA (NodeName, Start, End, ParentStart, Content)**

Each tuple in the BS-Containment-METADATA relation represents an XML node in the marked XML tree. The NodeName attribute represents the tag name of XML node in the marked XML tree and the Start, End, and ParentStart attributes correspond to the start, end, and parentStart values of each XML node in the BS-Containment labeling scheme. Moreover, the Start attribute serves as the primary key of the relation. Besides, the Content attribute represents the value of XML node if it is a leaf node in the marked XML tree. An instance of BS-Containment-METADATA relation for the marked XML tree illustrated in Fig. 3 is illustrated in Table 2. Note that the Content attribute of all the internal XML nodes as well as the root node is “Null”.

Moreover, the metadata of authorization nodes in the DCAM labeled using the BS-Containment labeling scheme is stored into the DCAM-METADATA relation. The schema of this relation is as follows:

**DCAM-METADATA (Start, End, Accessibility)**

Each tuple in the DCAM-METADATA relation represents an authorization node in the marked XML tree. The Start and End attributes correspond to the start and end values of authorization nodes in the marked XML tree. Moreover, the Start attribute serves as the primary key of the relation. Besides, the Accessibility attribute represents the accessibility of authorization nodes. An instance of DCAM-METADATA relation for the marked XML tree illustrated in Fig. 3 is shown in Table 3.
According to the definition of authorization node, the accessibility of an XML node is determined by finding its nearest authorization node in the marked XML tree. The definition of nearest authorization node of an XML node is as follows:

**Definition of Nearest Authorization Node:** An authorization node is called the nearest authorization node \(NAN_\alpha\) of node \(\alpha\) if it satisfies the following two conditions:

1. \(NAN_\alpha\) is an authorization node which can be node \(\alpha\) or one of its ancestor nodes;
2. No authorization node exists in the depth between the node \(\alpha\) and the node \(NAN_\alpha\).

The process of finding the nearest authorization node for each XML node can be efficiently done since the XML nodes as well as the authorization nodes in the marked XML tree are labeled using the BS-Containment labeling scheme. The algorithm of finding the nearest authorization node is shown in Fig. 8.

**Fig. 8. Algorithm of finding the nearest authorization node.**

**FindNearestAuthorizationNode (targetNode)**

*Input:* targetNode labeled by \((\text{start}_{\text{targetNode}}, \text{end}_{\text{targetNode}}, \text{parentStart}_{\text{targetNode}})\)

*Output:* the nearest authorization node \(NAN_{\text{targetNode}}\)

1. \(NAN_{\text{targetNode}} \leftarrow \text{Select } * \text{ From DCAM-METADATA Where (Start } \leq \text{start}_{\text{targetNode}} \text{ and End } \geq \text{end}_{\text{targetNode}}\);
Lemma 5.1: The algorithm illustrated in Fig. 8 correctly finds the nearest authorization node of each XML node in the marked XML tree.

Proof: In order to prove the correctness of the algorithm illustrated in Fig. 8, we must show that the algorithm satisfies the two conditions stated in the definition of nearest authorization node. Assume that we want to find the nearest authorization node of node $\alpha$. According to the first condition in the definition of nearest authorization node, the nearest authorization node of node $\alpha$ is either node $\alpha$ or one of its ancestor nodes. Since the authorization nodes in the DCAM are labeled using the BS-Containment labeling scheme, we can find all the ancestor nodes of node $\alpha$ using the following two conditions as presented in the where closure of the sub-query in the line 1: $(\text{Start} \leq \text{start}_\alpha)$ and $(\text{End} \geq \text{end}_\alpha)$. Since the nearest authorization node may be the node $\alpha$, we also add the operator $(\neq)$ in these two conditions. Therefore, the sub-query which finds the set of all authorization nodes of node $\alpha$ is as follows:

$$\text{Select } * \text{ From DCAM-METADATA Where } (\text{Start} \leq \text{start}_\alpha) \text{ and } (\text{End} \geq \text{end}_\alpha).$$

Moreover, by finding the authorization node with the maximum start value from the set of authorization nodes, we can satisfy the second condition in the definition of nearest authorization node since the nearest authorization node is an authorization node that the difference between its start value with the start value of node $\alpha$ is minimized. Therefore, the final sub-query which finds the start value of nearest authorization node of node $\alpha$ is as follows:

$$\text{Select Max (Start) From DCAM-METADATA Where } (\text{Start} \leq \text{start}_\alpha) \text{ and } (\text{End} \geq \text{end}_\alpha).$$

By using the final sub-query in the where closure of the SQL query illustrated in line 1, we can find the metadata of nearest authorization node of node $\alpha$. □

6. ACCELERATING THE PROCESS OF ACCESS AUTHORIZATION CHECKING

Assume that a set of XML nodes is retrieved by the XML query processor. Now, the accessibility of each retrieved XML node must be checked in order to decide whether the XML node should be returned to the subject or not. If we use the lookup method proposed in Section 5 to determine the accessibility of each XML nodes, it is necessary that the nearest authorization node of each retrieved XML node is found.

Example 6.1: As illustrated in Fig. 9, assume that $\{B_1, B_2\}$ be the set of authorization nodes and $\{N_1, N_2, \ldots, N_{200}\}$ be the set of XML nodes retrieved by the XML query processor. Using the lookup method proposed in Section 5, we need to find the nearest authorization nodes for the 200 XML nodes in order to check the accessibility of these XML nodes.

A solution to accelerate the process of access authorization checking is to reduce the number of access authorizations which must be automatically checked by the system
[4, 11, 19, 27]. This solution can be used by the DCAM when the access locality among the XML node in the marked XML tree is high.

Example 6.2: As shown in Fig. 9, the ratio of the number of retrieved XML nodes to the number of authorization nodes is 100 (100 = 200/2). It means that the access locality among the XML nodes in the marked XML tree is high. In Fig. 9, all the XML nodes from the node $N_i$ to the node $N_{100}$ have the same accessibility since the nearest authorization node for them is the node $B_1$. They form the set of unauthorized XML nodes. Moreover, the XML nodes from the node $N_{100}$ to the node $N_{200}$ have the same accessibility since the nearest authorization node for them is the node $B_2$. They form the set of authorized XML nodes.

We can define an interval using the BS-Containment labeling scheme for a set of XML nodes with the same accessibility. Using this interval, we are able to reduce the number of access authorizations which must be checked at runtime. In order to construct this interval, we define the nearest override authorization node as follows:

**Definition of Nearest Override Authorization Node:** Suppose that $NAN_\alpha$ is the nearest authorization node of node $\alpha$. An authorization node is called the nearest override authorization node $NOAN_\alpha$ of the nearest authorization node $NAN_\alpha$ if it satisfies the following two conditions:

1. $NOAN_\alpha$ is a descendant node of node $NAN_\alpha$ with different accessibility;
2. No authorization node exists between the node $\alpha$ and the node $NOAN_\alpha$ when the marked XML tree is traversed in the preorder.

The algorithm of finding the nearest override authorization node is shown in Fig. 10 and its correctness is proved by Lemma 6.1.

**Lemma 6.1:** The algorithm illustrated in Fig. 10 correctly finds the nearest override authorization node of each XML node in the marked XML tree.

**Proof:** In order to prove the correctness of the algorithm illustrated in Fig. 10, we must
show that the algorithm satisfies the two conditions stated in the definition of nearest override authorization node. Assume that we want to find the nearest override authorization node (NOAN) for the nearest authorization node (NAN) of node α. According to the first condition in the definition of nearest override authorization node, the nearest override authorization node of node NAN is one of descendant nodes of node NAN so that its accessibility is different with the accessibility of NAN. Since the authorization nodes in the DCAM are labeled using the BS-Containment labeling scheme, we can find all the descendant nodes of node NAN with the different accessibility using the following conditions as presented in the where closure of the sub-query in the line 1:

\[(\text{Start} > \text{start}_{NAN}) \text{ and } (\text{End} > \text{end}_{NAN}) \text{ and } (\text{Accessibility} \neq \text{Accessibility}_{NAN})\].

Moreover, by finding the authorization node with the minimum start value from the set of descendant nodes of node NAN as well as adding the condition (\text{Start} > \text{start}_{\alpha}), we can satisfy the second condition in the definition of nearest override authorization node since the nearest override authorization node is an authorization node that the difference between its start value with the start value of node α is minimized and appear after node α when the marked XML tree is traversed in preorder. Therefore, the final sub-query which finds the start value of nearest override authorization node of node NAN is as follows:

\[
\text{Select Min (Start) From DCAM-METADATA Where (Start > start}_{\alpha}) \text{ and (Start > start}_{NAN}) \text{ and } (\text{End} < \text{end}_{NAN}) \text{ and (Accessibility} \neq \text{Accessibility}_{NAN}));
\]

By using the final sub-query in the where closure of the SQL query illustrated in line 1, we can find the metadata of nearest override authorization node of node NAN.

\[
\text{FindNearestOverrideAuthorizationNode (targetNode, NANtargetNode) Input: targetNode and the nearest authorization node (NANtargetNode) of node targetNode Output: the nearest override authorization node NOAN}
\]

1. NOAN \(\leftarrow\) (Select * From DCAM-METADATA Where Start = (Select Min (Start) From DCAM-METADATA Where (Start > start}_{targetNode}) \text{ and (Start > start}_{NANtargetNode}) \text{ and } (\text{End} < \text{end}_{NANtargetNode}) \text{ and (Accessibility} \neq \text{Accessibility}_{NANtargetNode}));

Fig. 10. Algorithm of finding the nearest override authorization node.

**Definition of Interval:** Suppose that the nearest authorization node of the node α is \(NAN_\alpha\) and the nearest overriding authorization node of \(NAN_\alpha\) is \(NOAN_\alpha\). The interval of a set of XML nodes which have the same nearest authorization node as that of the node α is defined as follow:

\[
\text{Interval} = \begin{cases} 
{[\text{start}_{\alpha}, \text{start}_{NAN_\alpha})} & \text{if the NOAN_\alpha exists} \\
{[\text{start}_{\alpha}, \text{end}_{NAN_\alpha})} & \text{if the NOAN_\alpha does not exist}
\end{cases}
\]
Example 6.3: In Fig. 9, the nearest authorization node of node $N_1$ is the authorization node $B_1$, and the nearest override authorization node of node $B_1$ is the authorization node $B_2$. Therefore, the interval of a set of XML nodes which have the same accessibility as the node $N_1$ is $[start_{B_1}, start_{B_2})$. It means that all the XML nodes which are in this interval are inaccessible since the node $N_1$ is inaccessible. Thus, instead of finding a set of nearest authorization nodes for the 99 XML nodes from the node $N_1$ to node $N_{99}$, we can accelerate the process of access authorization checking at runtime using this method.

7. MAINTAINING THE DCAM IN DYNAMIC ENVIRONMENT

In dynamic environments, new XML nodes can be inserted into the XML tree and existing XML nodes can be deleted from the XML tree. The process of node insertion and node deletion may cause inconsistency in the XML database. For example, assume that an internal node (i.e. a node in the XML tree which has at least a child node) which is also an authorization node is deleted from the XML tree. Now, the question is how to determine the accessibility of a child node of the deleted node which is not the authorization node. If an authorization node is deleted from the XML tree, the metadata of that node must be deleted from the DCAM-METADATA relation. By deleting the authorization node from the DCAM-METADATA relation, we are not able to determine the accessibility of a child node of the deleted node which is not the authorization node. In this case, we need to do extra works to ensure the consistency of XML database and maintain the DCAM in a compact format.

On the other hand, in dynamic environment, the accessibility of XML nodes may be changed by the security administrators. By changing the accessibility of an XML node, the position of a large number of authorization nodes may be changed in the marked XML tree. In this case, the question is how to maintain the DCAM as compact as possible with minimum maintenance cost.

In this section, we define a set of constraints for the process of node insertion, the process of node deletion, and the process of changing the accessibility of XML nodes in order to ensure the consistency of XML database as well as to maintain the DCAM in a compact format in dynamic XML environment.

Constraint 7.1: The accessibility of a new inserted node is similar to the accessibility of its nearest authorization node in the marked XML tree.

Using the constraint 7.1, the process of node insertion is done without any maintenance cost on the DCAM. For example, in Fig. 3, the accessibility of the new inserted node $\alpha$ is similar to the accessibility of node $L$ since the nearest authorization node of node $\alpha$ is the node $L$.

Constraint 7.2: In the process of deleting a leaf node from the XML tree, if the deleted node is an authorization node, only the metadata of the deleted node must be deleted from the DCAM-METADATA relation otherwise do nothing.

Using the constraint 7.2, at worst case, the maintenance cost of the DCAM in the
process of leaf node deletion contains deleting a tuple from the DCAM-METADATA relation. For example in Fig. 3, if node $L$ is to be deleted from the XML tree, the metadata of node $L$ must be deleted from the DCAM-METADATA relation while if node $F$ is to be deleted from the XML tree, there is no need to delete a tuple from the DCAM-METADATA relation.

**Constraint 7.3:** In the process of deleting an internal node from the XML tree, if the deleted node is an authorization node, the metadata of its children nodes which are not authorization nodes must be inserted into the DCAM-METADATA relation and the metadata of the deleted node as well as its children nodes which are authorization nodes must be deleted from the DCAM-METADATA relation otherwise do nothing.

For example in Fig. 3, if the node $D$ is to be deleted from the XML tree, the metadata of node $H$ and node $I$ must be inserted into the DCAM-METADATA since they are not the authorization nodes and the metadata of node $D$ and node $G$ must be deleted from the DCAM-METADATA relation since the node $D$ is the deleted node and node $G$ is the child node of node $D$ which is also an authorization node.

**Constraint 7.4:** In the process of changing the accessibility of a leaf node by the security administrator, if the leaf node is an authorization node, the metadata of the leaf node must be deleted from the DCAM-METADATA relation otherwise the metadata of the leaf node must be inserted into the DCAM-METADATA relation.

For example, in Fig. 3, if the accessibility of leaf node $L$ is to be changed from “0” to “1” by the security administrator, the metadata of node $L$ must be deleted from the DCAM-METADATA relation while if the accessibility of leaf node $C$ is to be changed from “1” to “0”, the metadata of node $C$ must be inserted into the DCAM-METADATA relation.

**Constraint 7.5:** In the process of changing the accessibility of an internal node by the security administrator, if the internal node is an authorization node, the metadata of its children nodes which are not authorization nodes must be inserted into the DCAM-METADATA relation and the metadata of that internal node as well as its children nodes which are authorization nodes must be deleted from the DCAM-METADATA relation otherwise the metadata of its children nodes which are authorization nodes must be deleted from the DCAM-METADATA and the metadata of that internal node as well as the metadata of its children nodes which are not authorization nodes must be inserted into the DCAM-METADATA relation.

For example, in Fig. 3, if the accessibility of node $D$ is to be changed from “0” to “1” by the security administrator, the metadata of node $H$ and node $I$ must be inserted into the DCAM-METADATA relation and the metadata of node $D$ and node $G$ must be deleted from the DCAM-METADATA relation.

Note that we add the ParentStart attribute into the DCAM-METADATA relation due to the need of determining the Parent-Child relationship between the XML nodes during the process of checking the set of constraints presented in this section.
8. EXPERIMENTAL RESULTS

We have conducted several experiments to demonstrate how the use of DCAM can help to reduce the storage cost of the CAM and verify the efficiency of the lookup method as well as the efficiency of the method proposed in Section 6 to accelerate the process of checking the access authorizations. Moreover, we performed several experiments to show the efficiency of the DCAM for the process of XML updating.

All the experiments were conducted on a 2 GHz Pentium (IV) processor with 1 GB of RAM running Windows XP professional. In our experiments, we used the XMark dataset which was generated by the XML document generator xmlgen proposed by [40]. Table 4 shows the characteristics of the XMark dataset.

<table>
<thead>
<tr>
<th>Size (MB)</th>
<th>Number of Elements</th>
<th>Maximum Depth</th>
<th>Maximum Fan Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>17,132</td>
<td>12</td>
<td>255</td>
</tr>
<tr>
<td>5</td>
<td>75,397</td>
<td>12</td>
<td>1,147</td>
</tr>
<tr>
<td>11.5</td>
<td>167,865</td>
<td>12</td>
<td>2,550</td>
</tr>
<tr>
<td>50</td>
<td>720,800</td>
<td>12</td>
<td>11,002</td>
</tr>
</tbody>
</table>

8.1 Experiment on Storage Space

We compared the storage cost of the DCAM to the CAM in two different cases: when the access locality among the XML nodes is low and when the access locality among the XML nodes is high. Two metrics Total Number of XML Nodes Required to Be Stored and Total Number of Bits Required to Represent the Accessibility of XML Nodes were used to measure the storage cost in our experiments. We defined the total number of bits required to represent the accessibility of XML nodes as a metric in our experiments because the storage cost for the DCAM and the CAM is related to not only the total number of XML nodes required to be stored but also the total number of bits required to represent the accessibility of XML nodes based on the Properties 3.1 and 3.3.

8.1.1 Storage cost with low access locality

In order to simulate the case that the access locality among the XML nodes is low, we randomly and uniformly selected the XML nodes from the XMark dataset with the size 50MB as the accessible nodes. We controlled the accessibility of the XML nodes in this situation with the accessibility ratio. The accessibility ratio is the fraction of the XML nodes in the XML dataset which are accessible. We simulated this situation for 10 subjects when the accessibility ratio is varied from 10% to 90%.

Figs. 11 (a) and (b) illustrate the average total number of XML nodes required to be stored and the average total number of bits required to represent the accessibility of the XML nodes for 10 subjects when the access locality among the XML nodes is low.

From Fig. 11, it is clear that the total number of XML nodes required to be stored and the total number of bits required to represent the accessibility of the XML nodes in the DCAM and the CAM are related to the access locality among the XML nodes. When the accessibility ratio is 10% and 90%, the access locality among the XML nodes is higher and therefore, the total number of XML nodes required to be stored and the total...
number of bits required to represent the accessibility of the XML nodes in the DCAM and the CAM are smaller while in the case that the accessibility ratio is 50%, the access locality among the XML nodes is less and therefore, the total number of XML nodes required to be stored and the total number of bits required to represent the accessibility of the XML nodes in the DCAM and the CAM are larger. Besides, although the total number of XML nodes required to be stored in the DCAM is more than that in the CAM, but the total number of bits required to represent the accessibility of the XML nodes in the CAM is more than that in the DCAM. Therefore, the storage cost of the CAM is more than the storage cost of the DCAM in the case that the access locality among the XML nodes is low.

8.1.2 Storage cost with high access locality

In order to simulate the case that the access locality among the XML nodes is high, we randomly and uniformly selected the XML nodes from the XMark dataset with the size 50MB as the selected nodes and then labeled them as accessible or inaccessible. We simulated the access locality among the XML nodes by propagating the accessibility of the selected nodes to their descendants using the most specific override takes precedence policy [41]. Moreover, the root of the XML dataset was selected as the selected node to make sure all the XML nodes are either accessible or inaccessible. In this situation, two parameters namely the propagation ratio and the accessibility ratio were defined to control the access locality among the XML nodes. The propagation ratio determines the percentage of XML nodes that are randomly and uniformly selected from the XML dataset as the selected nodes while the accessibility ratio determines the percentage of selected nodes that are accessible. We set the propagation ratio to 2% to simulate the case that access locality among the XML nodes is high.

Figs. 12 (a) and (b) illustrate the average total number of XML nodes required to be stored and the average total number of bits required to represent the accessibility of the XML nodes in the DCAM and the CAM for 10 subjects when the access locality among the XML nodes is high.

Similar to Fig. 11, in Fig. 12, the total number of XML nodes required to be stored and the total number of bits required to represent the accessibility of the XML nodes in the DCAM and the CAM are related to the access locality among the XML nodes. When the accessibility ratio is 10% and 90%, the access locality among the XML nodes is higher and therefore, the total number of XML nodes required to be stored and the total number of bits required to represent the accessibility of the XML nodes in the DCAM and the CAM are smaller than in the case of low access locality.
number of bits required to represent the accessibility of the XML nodes in the DCAM and the CAM are smaller while in the case that the accessibility ratio is 50%, the access locality among the XML nodes is less and therefore, the total number of XML nodes required to be stored and the total number of bits required to represent the accessibility of the XML nodes in the DCAM and the CAM are larger. Besides, due to the high access locality in the case that the propagation ratio is 2%, the difference between the total number of XML nodes required to be stored in the DCAM and the CAM is much less compared to the case that the access locality is low.

8.1.3 Effect of diversity of access locality on storage cost

Fig. 13 illustrates the average total number of bits required to represent the accessibility of the XML nodes in the DCAM for 10 subjects when the access locality and the accessibility ratio are varied while the size of XML dataset is 50MB. As shown in Fig. 13, the average total number of bits required to represent the accessibility of the XML nodes in the DCAM is smaller when the access locality among the XML nodes is higher. It verifies that the high access locality has a positive effect on the reduction of storage cost in the DCAM.

8.1.2 Effect of number of subjects on storage cost

We performed the experiments under several situations in order to demonstrate the
effect of number of subjects on storage cost of the DCAM and the CAM. Since the results under different situations are similar, we only show the result for the following settings: the propagation ratio = 2%, the accessibility ratio = 30%, the XML dataset size = 50MB. Fig. 14 shows the total number of bits required to represent the accessibility of XML nodes in the DCAM and the CAM when the number of subjects is varied.

As shown in the Fig. 14, the growth rate of total number of bits required to represent the accessibility of XML nodes in the DCAM is less than that in the CAM. It means that the storage efficiency of the DCAM is better than the CAM in the multiple subject environments.

8.2 Experiment on Construction Process

We performed the experiments under several situations in order to compare the construction time of the DCAM to the CAM. Figs. 15 (a) and (b) illustrate the experimental results for two cases (a) when the access locality among the XML nodes is low and the accessibility ratio is 30%; (b) when the access locality among the XML nodes is high (the propagation ratio is 2%) and the accessibility ratio is 30%. We only show the results for these two cases since the other cases have similar results.

As shown in Figs. 15 (a) and (b), the average time to construct the DCAM and the CAM for 10 subjects is more when the dataset size is larger. The reason is that, the number of XML nodes in the marked XML tree increases as the size of dataset increases. Moreover, the growth rate of construction time in the DCAM is less than that in the
CAM. It means that the performance claims by the DCAM is better than the CAM.

8.3 Experiment on Accessibility Lookup

In order to verify the performance claims on the lookup method proposed in Section 5, we first randomly and uniformly selected 1,000 XML nodes from the XMark dataset with the size 50MB and then set the propagation ratio to 2% and 5% to perform the experiments with different settings. Figs. 16 (a) and (b) illustrate the results for the cases where the propagation ratio is 2% and 5%, respectively. As shown in Figs. 16 (a) and (b), the lookup method in the DCAM is more efficient than that in the CAM.

Fig. 16. Lookup Time (a) The Propagation Ratio = 2%; (b) The Propagation Ratio = 5%.

8.4 Experiment on Access Authorization Checking Process

As explained in Section 6, the proposed method to accelerate the process of checking the access authorizations can be used for a set of XML nodes retrieved from the XML query processor. Therefore, in order to simulate this situation, we first retrieved a set of XML nodes from the XMark dataset with the size 50MB and then compared the time to check the access authorizations in the DCAM using the method proposed in Section 6 (DCAM1), the DCAM using lookup method proposed in Section 5 (DCAM2), and the CAM using its own lookup method (CAM).

In our experiments, we used two sets of XML nodes which are retrieved at different levels of the XML tree as shown in Table 5.

<table>
<thead>
<tr>
<th>Query Name</th>
<th>Query Definition</th>
<th>Level of Retrieved Nodes in the XML tree</th>
<th>Number of Retrieved Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>//item</td>
<td>4</td>
<td>9,381</td>
</tr>
<tr>
<td>Q2</td>
<td>//mail</td>
<td>6</td>
<td>9,012</td>
</tr>
</tbody>
</table>

8.4.1 Effect of different methods on access authorization checking time

Figs. 17 (a) and (b) illustrate the results for the queries Q1 and Q2, respectively for the case that the propagation ratio is 2%, the XML dataset size is 50MB, and the accessibility ratio is varied. As shown in Figs. 17 (a) and (b), it is clear that the time to check
the access authorizations in the DCAM using the method proposed in Section 6 (DCAM1) is the least when the accessibility ratio is 10% and 90%. In the case that the accessibility ratio is 30%, 50% or 70%, the time to check the access authorizations in the DCAM using the lookup method proposed in Section 5 is the least. The reason is that, in the case that the access locality is low, the method proposed in Section 6 finds the nearest authorization node as well as the nearest override authorization node for many retrieved XML nodes. Finding the nearest override authorization nodes for many nodes reduces the efficiency of the method proposed in Section 6. However, the efficiency of the DCAM using the method proposed in Section 6 or using the lookup method proposed in Section 5 is better than the CAM using its own lookup method.

Fig. 17. Access authorization checking time (a) for the Query Q1; (b) for the Query Q2.

8.4.2 Effect of diversity of access locality on access authorization checking time

Figs. 18 (a) and (b) show the time to check the access authorizations in the DCAM using the method proposed in Section 6 for the queries Q1 and Q2, respectively when the XML dataset size is 50MB and the propagation ratio and the accessibility ratio are varied. As shown in Figs. 18 (a) and (b), the time to check the access authorizations in the DCAM when the propagation ratio is smaller is less. It means that the time to check the access authorizations with smaller propagation ratio has lesser overhead than that with larger propagation ratio. The reason is that the total number of access authorizations which is eliminated to be checked in the DCAM increases in smaller propagation ratio since the access locality among the XML nodes increases.

Fig. 18. Access authorization checking time for the DCAM (a) for the query Q1; (b) for the query Q2.
8.5 Experiment on Dynamic XML Updating

The efficiency of the BS-Containment labeling scheme for XML updating is demonstrated in [32]. Therefore, we did not repeat the experiments for it. We compared the maintenance cost of the DCAM using the set of constraints defined in Section 7 with the re-construction cost of the CAM in dynamic environment by performing several experiments. According to the constraint 7.1, the process of node insertion contains no maintenance cost for the DCAM. Therefore, we only conducted the experiments for the process of node deletion as well as the process of changing the accessibility of XML nodes.

8.5.1 Experiment on the process of node deletion

In order to verify the performance claims by the DCAM in the process of node deletion, we first randomly and uniformly selected 10 XML nodes from the XMark dataset with the size 50MB. Then, we deleted each XML node from the XML tree and measured the time to maintain the DCAM in compact format using the set of constraints defined in Section 7 as well as the time to re-construct the CAM. Fig. 19 illustrates the result when the propagation ratio is 2% and the accessibility ratio is varied.

As shown in Fig. 19, the time to maintain the DCAM in compact format is less than the time to re-construct the CAM when the XML nodes are deleted from the XML tree. It means that the DCAM is more efficient than the CAM to be used in dynamic XML environment.

8.5.2 Experiment on the process of updating the accessibility of XML nodes

In order to demonstrate the efficiency of DCAM in the process of changing the accessibility of XML nodes, we first randomly and uniformly selected 10 XML nodes from the XMark dataset with the size 50MB. Then, we changed the accessibility of each XML node and measured the time to maintain the DCAM in compact format using the set of constraints defined in Section 7 as well as the time to re-construct the CAM. Fig. 20 illustrates the result for the case that the propagation ratio is 2% and the accessibility ratio is varied.
87

Fig. 20. Maintenance cost versus accessibility ratio for the process of changing the accessibility of XML nodes.

As shown in Fig. 20, the time to maintain the DCAM in compact format is less than the time to re-construct the CAM when the accessibility of XML nodes is updated. It means that the DCAM is more suitable than the CAM to be used in dynamic XML environment.

9. DISCUSSION

In this section, we discuss about the possibility of constructing an Integrated DCAM for multiple subjects instead of constructing a DCAM for each subject. The process of constructing an Integrated DCAM (IDCAM) for multiple subjects is similar to the process of constructing the DCAM for a single subject.

Fig. 21 shows the marked XML trees corresponding to three subjects. Similar to Fig. 1, in Fig. 21, the gray rectangles represent accessibility of XML nodes to the subject while the white rectangles represent non-accessibility of XML nodes to the subject. Moreover, in Fig. 21, the authorization nodes are shown with $\star$.

As shown in Fig. 21, it is clear that the sub-set of authorization nodes for a subject is the same for other subjects. It means that the marked XML trees of different subjects have the authorization node similarity. For example, two authorization nodes $A$ and $D$ are the same in the DCAMs of the subjects 1, 2, and 3 as shown in Figs. 21 (a), (b), and (c). Therefore, by exploiting the authorization node similarity among the different subjects, we can integrate multiple DCAMs together. An example of the integrated marked XML tree for the three subjects is illustrated in Fig. 22 (a). As shown in the Fig. 22 (a), the set of authorization nodes (illustrated with $\star$) in the Integrated DCAM is the union of the authorization nodes in the DCAM of each subject.

**Property 9.1:** Let $S = \{S_1, S_2, \ldots, S_n\}$ be the set of subjects and $AN(DCAM_{s_i, a_j})$ be the set of authorization nodes in the DCAM of the subject $s_i$ and the action $a_j$. The set of authorization nodes in the IDCAM of action $a_j$ is $AN(IDCAM_{a_j}) = \bigcup_{i=1}^{n} AN(DCAM_{s_i, a_j})$ for $i = 1$ to $n$.

The total number of authorization nodes in the IDCAM depends on the authorization node similarity in the marked XML trees of different subjects. If the authorization node similarity in the marked XML trees of different subjects is high, the total number of authorization nodes in the IDCAM is low and vice versa.
Property 9.2: Let $|AN(\text{DCAM}_{s_i, a_j})|$ be the total number of authorization nodes in the DCAM of the subject $s_i$ and the action $a_j$ and $|AN(\text{IDCAM}_{a_j})|$ be the total number of authorization nodes in the IDCAM of action $a_j$. Therefore, we have: $1 \leq |AN(\text{IDCAM}_{a_j})| \leq \sum |AN(\text{DCAM}_{s_i, a_j})|$ for $i = 1$ to $n$.

For example, in Fig. 22 (a), the total number of authorization nodes in the IDCAM is “7” (“1” $\leq$ “7” $\leq$ “12”).

Property 9.3: Let $n$ be the total number of subjects. The total number of bits required to represent the accessibility of the authorization nodes in the IDCAM of action $a_j$ is $(|AN(\text{IDCAM}_{a_j})| \times n)$ bits.

For example, the total number of authorization nodes in Fig. 22 (a) is “7” and the total number of subjects is “3”, therefore, the total number of bits required to represent the accessibility of the authorization nodes in the IDCAM is “21” bits.

---

Fig. 21. The marked XML tree for three subjects.

Fig. 22. The IDCAM for three subjects.
Theorem 9.1: The order of the subjects does not affect on the total number of authorization nodes in the IDCAM of the action $a_j$.

Proof: We establish the proof by contradiction. Assume that the order of the subjects affects on the total number of authorization nodes in the IDCAM of the action $a_j$. Therefore, there are two integrated DCAMs IDCAM$_1$ and IDCAM$_2$ for the action $a_j$ with different number of authorization nodes ($|AN(IDCAM_1)| \neq |AN(IDCAM_2)|$). This is a contradiction based on the Property 9.1 since the set of authorization nodes in the IDCAM of action $a_j$ is the union of the authorization nodes in the DCAM of each subject.

For example, the order of subjects in Figs. 22 (a) and (b) is different but the set of authorization nodes as well as the total number of authorization nodes for them are the same.

Although we can easily construct an integrated DCAM (IDCAM) for a specific action and multiple subjects but the total number of bits required to represent the accessibility of XML nodes in the IDCAM is more than the total number of bits required to represent the accessibility of XML nodes in the DCAMs of different subjects. It means that the storage cost increases by integrating the multiple DCAMs into an IDCAM. For example, the total number of bits required to represent the accessibility of XML nodes in the IDCAM illustrated in Fig. 22 (a) is “21” bits while the total number of bits required to represent the accessibility of XML nodes in DCAMs illustrated in Fig. 21 is “12” bits ($4 + 3 + 5 = 12$).

9. CONCLUSION AND FUTURE WORKS

In this paper, we proposed a Dynamic Compressed Accessibility Map called DCAM which implements the accessibility map in a compact format by exploiting the access locality among the XML nodes in the marked XML tree. The advantage of the DCAM is its usability in dynamic XML environment where XML nodes are inserted into and deleted from the XML tree. Moreover, we devised an efficient lookup method to determine the accessibility of XML nodes by labeling the authorization nodes in the DCAM as well as the XML nodes in the marked XML tree with the BS-Containment labeling scheme. We also proposed an efficient method to accelerate the process of access authorization checking for the set of XML nodes retrieved from the XML query processor when the access locality among the XML nodes is high. In order to maintain the DCAM in compact format in dynamic XML environment, we defined a set of constraints for the process of XML updating. The experimental results demonstrated that the DCAM is more efficient than the CAM in both storage and time requirements even in static XML environment.

As a future work, we attempt to extend the usability of DCAM for the complex query language XQuery.

REFERENCES

1. T. Yu, D. Srivastava, L. V. S. Lakshmanan, and H. V. Jagadish, “A compressed ac-
querying XML data,” in Proceedings of Workshop on Secure Web Services, 2005, pp. 36-42.
31. M. Mirabi, H. Ibrahim, N.I. Udzir, and A. Mamat, “XML access control models and


Meghdad Mirabi obtained his Master and Ph.D degrees in Computer Science from the Universiti Putra Malaysia (UPM), Malaysia in 2009 and 2013, respectively. His research interests include XML data management, wireless information retrieval, and information security.
Hamidah Ibrahim is currently a Professor at the Faculty of Computer Science and Information Technology, Universiti Putra Malaysia. She obtained her Ph.D in Computer Science from the University of Wales Cardiff, UK in 1998. Her current research interests include databases (distributed, parallel, mobile, bio-medical, XML) focusing on issues related to integrity constraints checking, cache strategies, integration, access control, transaction processing, and query processing and optimization; data management in grid and knowledge-based systems.

Leila Fathi obtained her Master degree in Computer Science from the Universiti Putra Malaysia (UPM), Malaysia in 2013. Her research interests include XML data management in mobile wireless networks and XML security.

Nur Izura Udzir is an Associate Professor at the Faculty of Computer Science and Information Technology, Universiti Putra Malaysia (UPM) since 1998. She received her Bachelor of Computer Science (1996) and Master of Science (1998) from UPM, and her Ph.D in Computer Science from the University of York, UK (2006). She is a member of IEEE Computer Society. Her areas of specialization are access control, secure operating systems, intrusion detection systems, coordination models and languages, and distributed systems. She is currently the Leader of the Information Security Group at the faculty.

Ali Mamat is an Associate Professor at the Faculty of Computer Science and Information Technology, Universiti Putra Malaysia (UPM). He obtained his Ph.D degree in Computer Science from the University of Bradford, U.K. in 1992. His research interests include data management, XML and ontology engineering.